Reflections on Programming with Grid Toolkits

Emiliano Tramontana[1] and Ian Welch[2]

[1] tramontana@dmi.unict.it,
WWW home page: http://www.dmi.unict.it/~tramonta,
Dipartimento di Matematica e Informatica, Università di Catania,
Viale A. Doria, 6 - 95125 - Catania, Italy
[2] ian@mcs.vuw.ac,nz,
WWW home page: http://www.mcs.vuw.ac.nz/~ian,
School of Mathematical and Computing Sciences, Te Kura Pangarau, Rorohiko,
Victoria University of Wellington, Wellington, New Zealand

Abstract. Grid applications are fragile when changes to service implementations, non-functional properties or communication protocols take place. Moreover, developing Grid applications with current toolkits result in a tangling of toolkit-specific and application-specific code that makes maintenance and evolution difficult. This paper proposes solving these problems by using reflection to open up Grid toolkits, and to allow Grid applications to be developed as if they were centralised applications. This would allow changes to be handled dependably, and a clean separation between toolkit-specific and application-specific code.

1 Introduction

Grid computing is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations" [Fos01]. Virtual organisations (VOs) cover the spectrum from long-lived collaborations between static sets of organisations to short-term collaborations between dynamic sets of individuals. Grid applications are built out of heterogeneous resources offered by VOs that use a range of communication technologies to interoperate.

Grid toolkits aim to simplify the task of developing Grid applications out of Grid services. The toolkits automatically generate code for communication between clients and services but programmers must still add toolkit-specific code to both client and service implementations, particularly if non-functional properties such as security are to be implemented. This tangling of toolkit-specific code and application-specific code makes it difficult to maintain applications. For example, porting existing applications to new toolkits may require manual changes to client and service code.

As the resources comprising the VO may change while an application is running, Grid applications should be able to cope with dynamic changes such as changes to communication protocols, service interfaces or the arrival or departure of services, Grid toolkits cannot do this transparently. There is no support for switching communication protocols at runtime, and coping with the other changes requires explicit programming by the application developer.

We argue that these shortcomings of existing toolkits could be addressed by adopting work done on using reflection to treat distribution as a non-functional concern and to open up the implementation of middleware.

2 Features of the Globus Toolkit

We use the Globus Toolkit version 3.0 (GT3) as an example of a state-of-the-art Grid toolkit. GT3 is a next-generation implementation of the Globus Toolkit based on Open Grid Service Architecture (OGSA) mechanisms [Fos01]. The OGSA uses emerging web services to ease the task of building Grid programs. The next two sections describe the limitations of GT3 with respect to distribution transparency, and transparent implement of non-functional concerns.

2.1 Distribution Transparency

The GT3 toolkit [Glo03] can automatically generate stub and skeleton implementations from a Web Services Description Language (WSDL) [CCMW01] description. This provides a degree of distribution transparency but additional toolkit-specific code must be added to both the client and service implementation. At the client side, code must be written to explicitly bind an instance of a stub class before using it to access the remote service. At the service side, the service implementation must inherit from the skeleton class or provide some additional methods to allow the skeleton to delegate operations to the service implementation.

The current approach requires regeneration of the stub and skeleton code, and manual changes to source code whenever the interface to services change or what is a local resource is replaced by a service. Handling service arrival or departure is supported by web service protocols but requires explicit programming at the client side. Ideally these concerns should be transparently implemented. This would ease maintenance and evolution as once concern could be changed independently of the other.

2.2 Non-Functional Concerns

GT3 provides bindings that allow services to be hosted by a range of containers. These containers can transparently implement some non-functional concerns such as security. Containers can usually either only implement a fixed set of non-functional concerns or application-level concerns. Implementing non-functional concerns at the infrastructure level, for example changing the underlying communication protocol, cannot be done because new non-functional concerns are implemented by intercepting application-level messages.

Ideally, the toolkit should allow new non-functional concerns to be implemented at both the application and infrastructure-level. These should still be able to be declaratively specified for a service thereby allowing a clean separation between non-functional concerns and application code. This would aid

maintainability and evolution. Additionally, providing a facility to install or remove non-functional concerns at runtime would allow changes without stopping a running application.

Although non-functional concerns can be enforced transparently at the service side, implementing the complementary concerns at the client side requires the programmer to add toolkit-specific code to their program. For example, when using GT3, providing the security non-functional concern at the service side simply requires adding security configuration information to a deployment descriptor for the servce. However, implementing the other half of the security concern at the client side requires some code setting the appropriate properties for the service's remote proxy.

Ideally, there should be the same separation of concerns on the client side as the service side. As for the service side, these concerns should be able to be installed and removed dynamically. Furthermore, to remove the possibility that clients and services get out of synchronisation, there should be support for synchronously installing and removing concerns at both the client and service side.

3 Proposed Approach

The proposed approach aims at supporting developers building object-oriented applications without making applications tangled with distribution related concerns and without requiring programmers to change applications when an adaptation is required to consider new technologies.

There are two aspects to this approach. (1) Programs are developed in a centralised manner and transparently distributed. (2) An open implementation of GT3 is used that allows dynamic changes at runtime.

3.1 Centralised Development

The Addistant [TSCI01] system provides distributed execution of "legacy" Java bytecode. The definition of legacy is programs that were originally developed to be executed on a single Java virtual machine (JVM). The users of Addistant specify the host where instances of classes are allocated, and how remote references are implemented. Addistant automatically transforms the bytecode at load time and uses a special configuration file to separate the specification of class location etc. from the actual program implementation.

In order to further automate distributing a centralised Java application so as to choose the most appropriate host for each object, a reflective software architecture has been proposed [DSPT02]. In such an architecture, at load time a component analyses each application class and transforms it so that allocation of instances will be performed on the basis of the calculated class parameters and the run time conditions of hosts and network. The architecture facilitates the integration of additional allocation policies to be easily inserted to consider other specific needs of classes. For example, an allocation policy could be proposed to

match the needs of a class, in terms of remote services used, with the known web services, in order to find the most appropriate host for executing its instances. Moreover, through interception we can potentially change method invocations on the fly to enforce the syntax of the method allowing access to the web service.

Applying automatic bytecode transformation to the Grid environment would allow programmers to develop centralised versions of their programs and then transparently distribute them. This would avoid the need for inheritance or the manual coding necessary to support the delegation approach. In addition, should services change location then the change could be achieved by modifying the specification rather than the code. Furthermore, to make an application dynamically adaptable to changes of service location, an appropriate adaptation component can be transparently inserted into the application when transforming it into a distributed version. This component would dynamically check the location of a requested service and find its new location when necessary (i.e. when the service has migrated to a new host). Given that the Grid environment also supports resource brokering as a first-class concept then it would make sense to integrate this configuration with existing resource brokering technologies.

Because we are not primarily concerned with legacy code, i.e. where the source code is unavailable, then source-based transformation could be used. This could be useful, since the programmer could intervene to customise the code resulting from the automatic transformation. However, operating transformations on compiled code would support runtime dynamic adaptation. This would be focus of further investigation.

Automatically transforming a centralised application into a distributed one has two further benefits. Firstly, the selection of a primitive (i.e. socket, RMI, GridRPC [NMS+02], etc.) that makes distributed objects communicate can be chosen only when transforming the application, so as to fit the environment where the application is going to be deployed. This approach makes the original application free of remote communication primitives, thus an application is easier to develop and evolve. Moreover, when a new mechanism is available for a different distributed environment, only an adaptation of the transformation tool has to be performed. Secondly, additional features can be added at transformation time to consider the needs of the specific target environment. Thus, components that make the communication reliable or that perform resource allocation could be added both at the client and server side along with the support for communication.

3.2 Reflective Middleware

Ideally, the declarative approach supported by existing toolkits should be retained but it should be possible to easily extend toolkits to integrate new capabilities and support dynamic changes to non-functional properties. Here, we intend to draw upon the exisiting literature on reflective middleware. Reflective middleware can be defined as "a middleware system that provides inspection and adaptation of its behaviour through an appropriate causally connected representation" [Cou]. Proponents of this idea suggest that this middleware will

be able to be adapted to its environment and be able to cope with change. For example, an application deployed on a mobile device could use middleware that dynamically detected that the device was no longer plugged into an office network and could switch to using GSM for communications. This would happen transparently with no requirement for changing the application itself, what happens is that the implementation of the middleware itself is changed so the appropriate type of communications is used. Examples of reflective middleware are the DynamicTao [KRL+00], OpenORB [BCA+] or mCharm [Caz00].

The notion here is to open up the Grid toolkit in a principled way. The initial target would be the communication infrastructure. A key problem, especially if considering dynamic adaption, would be how to control the adaptation process. One way of doing this is to use the notion of adaptation policies, as is used for the K-Components framework [DC01]. In this framework an adaptation policy is specified using an extended interface description language. The policy is essentially a declarative language for writing reflective programs that can monitor and reconfigure programs by modifying the metalevel. Having a policy allows reasoning and validation of possible adaptations.

Another target would be the service container. Here the focus should be on an infrastructure for the server side that holds information about the current services that containers offer. This infrastructure would check at run time both the conditions of the containers and whether some service unavailable on a container is being requested. The aim of the infrastructure would be to dynamically and transparently transfer the requests to other containers providing the requested service, as appropriate.

The design of this infrastructure would be based upon the design lessons of existing reflective middleware. Providing first-class support for dynamic evolution of Grid applications would enhance the dependability of services because it allows clients requests to be redirected where they can be honoured, thus avoiding failures on the client side.

4 Related Work

The most closely related work in the Grid community is Othman et. al. [ODDG] who use OpenJava to simplify the implementation of an adaptive resource broker. The adaptive resource broker allows running jobs to be suspended and migrated to other hosts for execution in order to satisfy a required quality of service. Our approach differs in that it considers the goals of making Grid toolkits easier to use and supports dynamic adapation of non-functional concerns.

5 Conclusions

Existing Grid toolkits ease the job of the programmer but could be improved by removing tangling between application code and toolkit code, and allow dynamic installation and removal of non-functional concerns at both the application and infrastructure level. In this paper we have identified related work that applies

reflection to distributed systems for similar purposes. We propose extending this work to develop an open Grid toolkit that hides distribution and allows a programmer to develop an application as if it was centralised rather than distributed.

References

[BCA+] G. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T. Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski. The design and implementation of open orb 2. IEEE Distrib. Syst. Online 2, 6 (Sept. 2001); see computer.org/dsonline.

[Caz00] Walter Cazzola. *Communication-Oriented Reflection: A Way to Open Up the RMI Mechanism*. PhD thesis, Università degli Studi di Milano, Italy, 2000.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (wsdl). Technical Report Note 15, 2001, W3C, 2001. `http://www.w3.org/TR/wsdl`.

[Cou] G. Coulson. What is reflective middleware? IEEE Distrib. Syst. Online 2, 8 (Dec. 2001); see computer.org/dsonline.

[DC01] Jim Dowling and Vinny Cahill. The k-component architecture meta-model for self-adaptive software. In *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, pages 81–88. Springer-Verlag, 2001.

[DSPT02] Antonella Di Stefano, Giuseppe Pappalardo, and Emiliano Tramontana. Introducing Distribution into Applications: a Reflective Approach for Transparency and Dynamic Fine-Grained Object Allocation. In *Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC'02)*, Taormina, Italy, 2002.

[Fos01] I. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, page 6. IEEE Computer Society, 2001.

[Glo03] Globus Project. *Java Programmer's Guide Core Framework*. `http://www.globus.org`, Mar 2003. `http://www-unix.globus.org/toolkit/3.0/ogsa/docs/java_programmers_guide%.html`, last update 09/03/2003, last access 31/03/2004.

[KRL+00] Fabio Kon, Manuel Romàn, Ping Liu, Jina Mao, Tomonori Yamane, Claudio Magalhaes, and Roy H. Campbell. Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In *IFIP/ACM International Conference on Distributed systems platforms*, pages 121–143. Springer-Verlag New York, Inc., 2000.

[NMS+02] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. Gridrpc: A remote procedure call api for grid computing. `http://www.eece.unm.edu/~apm/docs/APM_GridRPC_0702.pdf`, 2002.

[ODDG] Abdulla Othman, Peter Dew, Karim Djemame, and Iain Gourlay. Adaptive grid resource brokering. in preparation.

[TSCI01] Michiaki Tatsubori, Toshiyuki Sasaki, Shigeru Chiba, and Kozo Itano. A bytecode translator for distributed execution of "legacy" java software. In *Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 236–255. Springer-Verlag, 2001.